PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 15/16 — A1

(11) International Publication Number: WO 98/26355

(43) International Publication Date: 18 June 1998 (18.06.98)

(21) International Application Number: PCT/AU97/00840

(22) International Filing Date: 10 December 1997 (10.12.97)

(30) Priority Data:
PO 4188 12 December 1996 (12.12.96) AU

(71) Applicant *(for all designated States except US)*: QUANTUM NETWORKS PTY. LTD. [AU/AU]; 20B Weeden Road, Artarmon, NSW 2064 (AU).

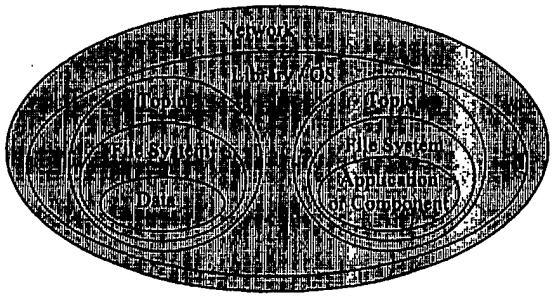
(72) Inventor; and
(75) Inventor/Applicant *(for US only)*: BROCKHURST, Russell [US/AU]; 20B Weeden Road, Artarmon, NSW 2064 (AU).

(74) Agent: F.B. RICE & CO.; 605 Darling Street, Balmain, NSW 2041 (AU).

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, GM, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, US, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published
With international search report.

(54) Title: A DISTRIBUTED OPERATING SYSTEM



(57) Abstract

A distributed operating system for a network, comprising a collection of libraries each of which exists at some physical location and is associated with a collection of instances of different topics. Each topic is an encapsulated set of system resources and has a file system containing applications, databases, software components, or any other collection of logically related files. Only one instance of a topic may exist in any given library at any given time. Instances of the same topic may exist in different libraries. An Addressing Standard which requires a first and second globally unique identifier is applied to the libraries and topics. The first globally unique identifier is used to identify each library. The second globally unique identifier is used to identify each topic. Every instance of a Topic will have the same topic identifier no matter which library it is in. Communications protocols exist outside the libraries to enable interaction between the applications, software components, and documents stored as Topics in different libraries.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece			TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	NZ	New Zealand		
CM	Cameroon			PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

"A distributed operating system"

Technical Field

This invention concerns a distributed operating system. Operating systems are the master control programs that run computers. They are an important component of a computer system, because they set the standards for the application programs that run in it. All programs must "talk to" the operating system.

Background Art

One of the central features of the architecture underlying most modern computer systems is that each Operating System defines a single name space for the data and applications managed by the system. For example, the name space defined by an operating system would include a File System or Directory Structure used to locate files, the identifiers for open files, common software components, and other network resources.

In such systems, the file is the basic unit of storage for information. A specific file either contains a unit of data or it contains a function (an application or a software component). For data files, the Operating System associates an appropriate application with each file so that the contents of the file can be viewed or updated.

Applications are built by integrating sets of Object Oriented software components. These Object Oriented software components may be custom built for a specific application or designed for use as a common component in many different applications.

The Compound File was developed so that the various Object Oriented components within an application could read and write data independently within a single file. While the Compound File is an effective means for storing the various objects (units of data) that make up a unit of information (a document), it can be difficult and time consuming to distribute a large Compound File over a network. For this reason, industry standards (such as HTML and Java) that build Internet (or Intranet) documents by breaking them down into sets of smaller files are increasing in popularity.

There is no consistent methodology within known Operating Systems for identifying collections of logically related files. This means that the Operating System cannot identify or manage collections of information that span multiple files such as a group of Internet (or Intranet) documents, a

common software component (including associated executable modules, templates, samples and related documentation) or a large database.

A physical address, called a Path, is used to locate files in a Directory Structure. The path is used for two conflicting purposes: to locate information, and to organise information. Files must be moved to reorganise information within a Directory Structure, and moving files changes the Paths that applications use to locate information.

Because each Operating System defines a single name space, it is impossible to predict if a specific name will be in-use within an operating system at any given point in time. This means that the names associated with specific system resources will vary from location to location and from time to time. For example, the installation processes defined for most applications allow the end user to choose the directory that will contain any associated files.

Because the names associated with files have no special meaning, Operating Systems cannot understand how information has been distributed across a network. This means that complex applications must be developed to manage the distribution of data, software components, and applications across networks.

A Local Area Network (LAN) extends the individual File Systems on a group of computers to include the shared File Systems on one of more LAN servers. Files on the LAN servers can be viewed and updated from any of the computers connected to the LAN. The LAN represents a logical evolutionary step from the stand-alone computer system as it extends the name space defined by an operating system to include selected resources on the LAN servers.

The basic problem with the LAN is that each of the computers connected to the LAN define different name spaces. A new name must be constructed in order to map the resources on one computer into the name space defined for another computer. A LAN can be very difficult to maintain if each workstation defines a different name for a set of common network resources. The solution to this problem is to define broad corporate standards for the configuration of LAN workstations. Maintaining these standards involves a great deal of corporate overhead and often frustrates end users. A more severe limitation of the LAN is that corporate standards can only be effectively enforced within a relatively small workgroup.

The World Wide Web, WWW, as a tool for global communication, owes its success to the use of an addressing mechanism based on a single globally unique identifier called a "Server Domain Name". Each Server Domain Name defines a distinct name space for a set of Internet (or Intranet) resources. A Server Domain Name provides a map to the name space defined by a stand-alone computer (or a LAN) connected to the Internet (or Intranet). The addressing mechanism used on the Internet (or Intranet) is called a Universal Resource Locater (URL) and is constructed by concatenating a Server Domain Name with a Path.

The Internet (or Intranet) represents a logical evolutionary step from the LAN since it could be defined as a LAN that uses a consistent global standard for naming network resources. The URL is an effective tool for information exchange if each unit of information exists exclusively within the name space defined by a single Server Domain Name, and the information associated with each Server Domain Name is not re-organised. Everyone that uses the Internet knows that it is already littered with out-of-date links even though it has only been in use for a few years. Further, the URL does not provide any mechanism for recognising information that is duplicated across multiple Server Domains. As with LAN's, complex (and specialised) applications must be implemented to manage the distribution of information across an Internet (or Intranet).

One of the objectives of the Object Oriented software development methodology is to define a set of standard software components that can be integrated to form complex applications. The idea is to bring to the software industry the same sort of modular engineering discipline that has been the foundation of the computer hardware industry for many years. One of the fundamental barriers to the realisation of this goal for the software industry is the fact that modern Operating Systems do not use consistent naming conventions. As a result, there can be no reliable method for finding a specific software component within a Directory Structure.

Microsoft's solution to this problem under Windows 95 and Windows NT is to build a System Registry. When a software component is installed, various entries must be made in the System Registry so that the component can be located and activated. When the component is removed from the system, the various entries in the System Registry must also be removed. A similar approach is used by the Common Object Request Broken

Architecture (CORBA) defined by the Object Management Group (OMG). This approach has several limitations when used to build complex distributed systems.

Because the System Registry does not understand the various registry entries that it maintains, it must rely on timely updates by the various software components and applications that are installed within the Operating System. While this approach is effective for Operating Systems that maintain a relatively static configuration, it is impractical for Operating Systems that must adapt to a large and continuously changing network environment.

The Windows 95 System Registry uses a single globally unique identifier, called a "CLSID", to locate common software components. The CLSID's used in the Windows 95 System Registry has the following limitations:

CLSID's are only defined for software components. Since data files are not able to create or maintain entries in the System Registry there is no way to use the System Registry to locate data files.

The System Registry acts as an addressing mechanism by associating a CLSID for a software component with a resource name (such as a Path) within an Operating System's name space. The System Registry is of limited use for building distributed systems since the addressing mechanism it defines is only useful within the scope of an individual Operating System. Even if you could read the entries in the System Registry defined for another Operating System, the results would not be meaningful.

A CLSID identifies the top-level object within a software component, so that an instance of the object can be instantiated. While the CLSID provides a method for locating and instantiating a component, it does not identify all of the resources (mostly files) that make up the component. For this reason, the CLSID is of limited use for managing the distribution of information across a network.

Microsoft's OLE and ActiveX protocols are the methodologies that Microsoft has defined for the integration of software components. Both protocols are based on Microsoft's Component Object Model (called "COM"), the OLE protocol has been optimised for the development of functions and applications that operate within a single Operating System, while the

ActiveX protocol has been optimised for the development of Internet (or Intranet) sites.

The OLE and ActiveX protocols support two basic services: a set of functions that allow an application to activate a software component, and a set of functions that allow applications to create, read, and update compound files. These protocols use the System Registry to locate individual software components, and are therefore constrained by the limitations of the System Registry as previously described, the protocols are also very complex.

The OLE and ActiveX protocols perpetuate the view of a network as the architectural equivalent of a large Operating System. Microsoft's notion of Local/Remote Transparency is essentially a methodology for mapping a set of distributed resources into an Operating System's name space. COM assumes that every component of the system exists in one and only one location on the network, and that applications are built by connecting together these distributed components.

Database technologies, including those offered by Oracle, IBM, Informix, and Sybase, consist of a set of files and a common software component. The common software component implements a specific database access method that allows other applications to access the information that has been stored in the files. One way to describe a database is as an encapsulated name space for a collection of information. The success of the Relational Database can largely be attributed to the fact that the relational access method allows every piece of data stored in the database to be used as a name for other data.

However, modern database technologies have several limitations, such as before a database can be used, the structure of the database (data types and indices) must be defined. Since defining the structure of a database requires a good deal of time and technical expertise, it can only be cost justified for high volume and repetitive business activities. Also, since each database defines a custom name space, anyone that wants to use a database must first understand how it is structured.

Databases that are defined independently will implement distinct and possibly incompatible name spaces. To solve this problem, corporations are encouraged to develop additional corporate standards for information management.

A Lotus Notes (recently renamed as "Lotus Domino") network consists of a Lotus Notes server and a number of clients. Both the server and the clients must run the proprietary software and databases can be stored either on the client or the server machine. Lotus Notes provides an environment in which groups of individuals can share information with each other. To view or edit a document within this environment, the user just points and clicks. Access to documents not created in the Notes environment is limited to applications supporting Object Linking and Embedding. Notes does, however, provide a front end to some of the more common databases but this facility is restricted to certain platforms.

The Lotus Notes environment is a closed, categorised and indexed reference to organisational information. It allows individuals to work as part of a team with provision for multi-user access to single documents and co-authoring facilities. Each database icon represents a collection of logically related documents sorted by some key item. These documents are added by individuals who wish to share information with other users within the organisation.

For individuals and groups within an organisation, Lotus Notes provides visibility of information and the opportunity to comment, edit and update documents. It also provides a facility for logically grouping documents. It does, however, tend to restrict users to organisational networks.

Lotus Notes and most other workgroup systems are essentially database technologies that have been designed to support a specific suite of workgroup applications. While these workgroup applications support an impressive range of features, they cannot overcome the limitations of their underlying database technologies: most workgroup systems are expensive to implement and maintain. Because of the associated costs, most workgroup systems can only be cost justified for use in support of high-volume and repetitive business activities.

Several existing products, including Intel Corp's LANDesk, automate the process of distributing software across LAN's (and in some cases across WAN's). For these systems to work effectively, still more corporate standards must be defined for the configuration of LAN workstations. The network management tools are then configured to manage the distribution of the applications that are included within the corporate standard.

Network management tools essentially automate the otherwise labour intensive process of visiting each workstation to install software updates. While these tools reduce the total cost associated with software distribution, they do not overcome the other limitations of the traditional software distribution process. For instance, Network management tools must be installed and managed by a central IT support group, and the central support group must maintain a hardware and software inventory for each of the workstations attached to the LAN. Because of the costs associated with the central support group, the network management tools can only be applied cost effectively for applications that are widely used throughout a corporation.

Network management tools don't have any unique knowledge concerning the applications they are distributing. The tools simply copy the updated files that make up the application and then make selected updates to any associated configuration files. Network management tools cannot support applications that require a highly customised distribution process.

Network management tools are only designed to distribute information that is installed on a large number of workstations, and that is static over time. Network management tools cannot be used to distribute documents, since documents are created and updated at unpredictable places and times across a network.

In summary, various applications have been described in terms of how they define names and how they distribute data. Most modern applications create islands of automation by defining custom names to meet specific business needs. Many layers of complexity have been added to the modern Operating System in order to integrate and reconcile the various names defined by different applications, and by different operating systems.

Summary of the Invention

The invention, as currently envisaged, provides a distributed operating system for a network, comprising a collection of libraries each of which exists at some physical location and is associated with a collection of instances of different topics. Each topic is an encapsulated set of system resources and has a file system containing applications, databases, software components, or any other collection of logically related files. Only one instance of a topic may exist in any given library at any given time. Instances of the same topic may exist in different libraries. An Addressing

Standard which requires a first and second globally unique identifier is applied to the libraries and topics. The first globally unique identifier is used to identify each library. The second globally unique identifier is used to identify each topic. Every instance of a Topic will have the same topic identifier no matter which library it is in. Communications protocols exist outside the libraries to enable interaction between the applications, software components, and documents stored as Topics in different libraries.

The principle advantage is that information, such as documents and common software components, produced independently can be combined to form integrated systems without the need for complex address translations.

The invention also allows for the identity of information to be retained as it is copied from one system to another. The ability to recognise how information has been distributed across a network will dramatically reduce the complexity of network maintenance activities such as software distribution and inventory control. The invention will also allow for the development of a new class of applications that mutate in useful ways as they are distributed.

A topic may be a document. Each instance of a document may contain the latest version of the document and might contain one or more older versions of the document. Updates to the various instances of the document are distributed as required.

A topic may be an application (or a software component). Each instance of an application contains a complete copy of the application. Updates to the instances of each application are distributed as required.

A topic may be a forum. One instance of a forum contains a complete copy of the forum. The other instances allow end users to review and add comments to the forum across a Network.

A topic may be a database. As with forums, one instance of the Topic contains a complete copy of the database while the other instances allow end users to access the database across a Network.

A topic may be an image archive. The various instances of the Topic manages the collection and distribution of document images for a project. Incremental updates are distributed to each instance of the Image Archive as required.

Each application is encapsulated within a Topic. To use a common software component, an application may use a set of Library services to

reference the appropriate resources within the Topic containing the software component. If the Library determines that the referenced Topic is located within another Library, then a similar set of Network services may be used to forward the request to the remote Library. Using this process, neither the application nor the software component needs to make any special provisions to support remote Topic references.

Since the Topic ID is a globally unique identifier for a distributed application, an application may let the Library (or Network) search for an instance of a specific Topic ID. If an instance of the database is available within the local Library, then the local instance may be used to satisfy the reference. If the local instance of the database cannot satisfy the request directly, then the local instance may act as a proxy for a remote instance of the database. If a local instance of the Topic cannot be found, then the Library may search through an ordered list of corporate Libraries to find an appropriate remote instance of the database.

Topics may be organised into hierarchical structures with a root topic and descendant topics. Libraries may organise Topics into any structure (hierarchical, matrix, and so on) since the Addressing Standard does not rely on the Library structure to locate Topics. A Library could even support multiple organisational structures simultaneously in order to offer various end users a customised "view" of the Library.

A particular end user's view of a Library could exclude any Topics for which the end user did not have a sufficient level of access authority. Separate access levels could be defined for addressing Topics and for viewing organisational structures. While a Topic containing a software component might not appear in an end user's view of a Library, the end user might still be allowed to use an application that referenced the software component.

There are two basic methods for managing versions. Versions may be created by copying Topics, or version control may be implemented within Topics. When a new version is created by copying a Topic, dependent relationships may be defined between the various versions of a Topic to simplify the associated coordination effort.

Alternatively, a Topic may contain a document with version control. A specific version of the document may be contained within a single compound file, or a single sub-directory. As new versions of the document

are created they may be stored in additional compound files or sub-directories within the same Topic. The Topic may also be used to consolidate and store any comments submitted with respect to the various versions of the document. The distribution of such a Topic may create a new instance (or update an existing instance) containing the latest version of the document, the older versions of the document, and any associated comments.

Version control for software components may be handled differently. A software development environment may be set up as a Library that generates new Topics to contain new software components and applications. Multiple versions of each software component (or application) may then be supported within each individual Topic. The distribution of such a Topic may create a new instance (or upgrade an existing instance) to contain the latest version of the software component (or application). Using this approach, all of the source code, executable modules, documentation, and installation procedures for a specific software component may be encapsulated within a single Topic. This approach simplifies the development of test environments and improves component reliability since the Topic ID and name associated with each software component may be constant from its first implementation through to its distribution.

Libraries may identify Topics either as "Trusted Topics" or "Suspect Topics". A Topic may be identified as a Trusted Topic if it was distributed from a Trusted Topic on a Trusted Library (or otherwise identified as a Trusted Topic by an authorised end user). Each Library may maintain a short list of Trusted Libraries from which it accepts the distribution of trusted applications and software components. The set of Trusted Libraries defined by a Library is the Trusted Domain for the Library. Any Topic that was not distributed from a Trusted Topic on a Trusted Library is identified as a Suspect Topic. A Suspect Topic is allowed to modify the resources contained within its own name space and within the name spaces of any Topics that list the Suspect Topic as a Named Dependent. However, the Suspect Topic is not allowed to update the resources contained in any other Topic. These restrictions effectively minimise the damage done to a Library or to a Network by a malicious Topic.

In addition to the resources encapsulated within topics, each topic can implement a set of functions, called "Methods", to control its own

distribution. It is not necessary for all of the Topic Methods be defined for every Topic. All of the methods and Library services used by a Topic are executed within the context of that specific Topic. Topics may inherit Library functions (and Operating System functions) from their associated Libraries.

To copy a Topic, the Copy Topic method defined for the Topic is called. The Copy Topic method creates a new Topic and initialises the contents of the new Topic. Finally, a method defined for the new Topic is called to complete the initialisation process.

To move a Topic within the active view, the Move Topic method implemented for the Topic is called. The Move Topic method removes the Topic as a direct descendent of its current ancestor, and installs the Topic as a direct descendent of a second Topic. A method is then called for all of the Topics that were moved. Note that when a Topic is moved, all of its descendants are also moved.

To delete a Topic, the DeleteTopic method implemented for the Topic's ancestor is called. Topics do not delete themselves because many Operating Systems do not allow a function to delete the file from which the function was loaded.

The Open Topic method is used to activate the application or software component that is associated with a Topic.

The Publish Topic method is used to distribute a Topic to another Library. The Publish Topic method assumes that the original instance of the Topic should remain on the source Library and that a copy should be distributed to the target Library. The PublishTopic method assumes that the PublishTopic methods for each Named Dependent have already been executed. The PublishTopic method will check the target Library to see if an instance of the Topic already exists. If an instance of the Topic already exists on the target Library then the BeforePublishTopic method implemented for the existing instance is called, and the contents of the existing Topic are updated. If an instance of the Topic cannot be found on the target Library, then an instance of the Topic is added to the target Library.

The Restore Topic method is identical to the Publish Topic method except that it restores a Topic to a previous version instead of updating the Topic to a more recent version. The Restore Topic method is designed to

work with the Publish Topic method so that Topics may be backed-up and recovered by "publishing" the Topics to another library and then "restoring" them as necessary. For example, if the original instance of a Topic were corrupted or deleted, the Restore Topic method could be used to regenerate the original instance of the Topic from a published instance. The Restore Topic method will not be supported when a published instance of a Topic contains insufficient information to regenerate the original instance.

The Update Topic method is used to control the automatic distribution of updates for Topics. When the Update Topic method is executed, the Topic checks to see if an update is available for distribution. If an update is available, the location of the instance of the Topic containing the update is returned so that the Publish Topic method may be executed to complete the update process.

The Route Topic method is used to distribute a Topic to another Library. The Route Topic method assumes that the original instance of the Topic should be moved to the target Library, and a copy of the Topic should be left on the source Library. To route a Topic, the RouteTopic method implemented for the Topic is called. The RouteTopic method assumes that the PublishTopic methods for each Named Dependent have already been executed. The RouteTopic method will check the target Library to see if an instance of the Topic already exists. If an instance of the Topic already exists on the target Library then the BeforeRouteTopic method implemented for the existing instance is called, and the contents of the existing Topic are updated. If an instance of the Topic cannot be found on the target Library, then an instance of the Topic is added to the target Library.

A global address is defined for each instance of a Topic as the combination of a Library ID, and a Topic ID. In addition to these global addresses, addresses may be defined in two other basic formats: Relative Addresses and Named Dependents. Compound Addresses are formed by chaining together global addresses, Relative Addresses and Named Dependents.

Each resource within a Topic has a name that is only valid within the context (or name space) defined by the Topic. Resources in other Topics are referenced by using the global address (or compound address) for the Topic as a prefix.

A Relative Address may be used by a Topic to identify other topics in the active view. Note that there is no way to define a Relative Address for a direct descendent since there is generally more then one direct descendent defined for each Topic.

A Topic is said to be dependent on a second Topic if the second Topic is defined by the first Topic as a Named Dependent, where a Named Dependent is a relationship between a name and a compound address. The Compound Addresses identified as Named Dependents describe a set of Topics on which the active Topic is dependent. The name associated with a Named Dependent may be used as an address.

A Tag is a relationship between a name and a short text phrase. The following standard Tags may be defined for all Topics:

Name - the name of the Topic.

<Compound Address> - the name of the Topic.

Author - the name of the individual that produced the Topic.

Address - the mailing address for the Author.

E-mail Address - the e-mail address for the Author.

Company - the Company that developed the Topic.

A Named File is a relationship between a name and a path. When the invention is implemented on top of a legacy system, there may be a need to include one or more files within a Topic without actually moving the files. In this regard, a Named File identifies a file that is encapsulated within a Topic but has not yet been embedded within the Topic's File System. Copies of any Named Files are embedded within the remote instances of the Topic, when the Topic is Published or Routed. Named Files are also used to generate HTML links for encapsulated files. For example, an HTML link will be generated for a text file that has been identified as a Named File so that the contents of the file may be viewed or edited through an Internet or Intranet.

Embodiments of the invention are particularly useful in supporting the automation of high-value and strategic projects which are typically one-off efforts of a relatively short duration. Such embodiments may include an Intranet publishing tool to support the automation of Outsourcing Sales Efforts and Due Diligence Studies for Corporate Acquisitions.

Brief Description of the Drawings

An example of the invention will now be described with reference to the accompanying drawings, in which:

Figure 1 is an organisational diagram of a Network embodying the present invention;

Figure 2 is an organisational diagram showing the relationship between libraries and topics;

Figure 3 is an organisational diagram showing a distributed topic, and

Figure 4 is an organisational diagram showing a hierarchical view of a library.

Best Modes for Carrying out the Invention

Referring first to figure 1, a Network 1 contains a number of Operating Systems, or Libraries 2, which in turn contain a number of encapsulated sets of system resources, or Topics 3. Each individual Topic 3 contains a File System 4, and the File Systems 4 may contain an application, a database, a software component, or any other collection of logically related files 5. The network 1 supports a set of communications protocols so that Topics 3 in different Libraries 2 can interact. The name spaces defined by the Libraries 2 is limited to the Topics 3.

At any given point in time, each Library 2 exists at a specific physical location on the Network 1 and is assigned a unique global identifier, a "Library ID". This identifier could be a Server Domain Name defined for the Internet (or for an Intranet), or it could be a random number that has a very high probability of being unique (such as the Open Software Foundation's UUID or Microsoft's GUID).

Each distinct Topic on the Network is also assigned a globally unique identifier, or "Topic ID". Topics that appear in more than one Library, are said to have multiple "instances". The various instances of a Topic will always be assigned the same Topic ID. Topic ID's could be defined as UUID's or as GUID's, or could be built from the Library ID of the Library in which the first instance of the Topic was created. The Addressing Standard is defined by concatenating a Library ID with a Topic ID.

Another way to define the Addressing Standard is to describe a Network 1 as two distinct sets: a set of Libraries 2, and a set of Topics 3, as shown in Figure 2. A Network is constructed by associating a set of Topics 3

with each Library 2 . Each Library 2 is then constructed by adding instances of the associated Topics 2 to the Library. Note that this approach clearly shows that two instances of the same Topic cannot appear in a single Library. This is an acceptable restriction because it really doesn't make sense to have two copies of an application, a software component, or a document in a single Library. Also, new Topics can be created by copying an instance of an existing Topic as required.

The instances of a Topic can easily be located (since they are all assigned identical Topic ID's), and may be integrated to form a distributed application 6, as shown in Figure 3.

It should be noted that an instance of a Topic is not necessarily an identical copy of a Topic. The content and behaviour of each instance of a Topic will vary for different Topics. For example, an Intranet publishing tool could include the following Topics:

- **Documents** - Each instance of a document contains the latest version of the document and might contain one or more older versions of the document. Updates to the various instances of the document are distributed as required.
- **Applications** (or software components) - Each instance of an application contains a complete copy of the application. Updates to the instances of each application are distributed as required.
- **Forums** - One instance of a forum contains a complete copy of the forum. The other instances allow an end user to review and add comments to the forum across a Network.
- **Databases** - As with forums, one instance of the Topic contains a complete copy of the database while the other instances allow an end user to access the database across a Network.
- **Image Archives** - The various instances of the Topic manage the collection and distribution of document images for a project. Incremental updates are distributed to each instance of the Image Archive as required.

Dependent relationships can be defined for each Topic. For example, a document that was created using a specific word processing application would ordinarily be dependent on that word processing application. Through the use of dependent relationships, the software

installed in a Library can be kept up-to-date and reconfigured as required to support the activities of an end user.

Incremental Implementation

Libraries and Topics may be built as a layer on top of existing, or "legacy", Operating Systems and applications. Initially applications will be implemented that make the greatest possible use of the features of the invention, additional functions would later be implemented using the invention and the dependence on legacy systems would be reduced.

The ability to implement embodiments of the invention as a layer on top of legacy systems is made possible by the definition of a Topic as an encapsulated set of system resources. In effect the invention may be implemented by partitioning the resources managed by the legacy systems into a set of Topics.

Applications implemented using the invention will reference resources by using the Addressing Standard. When an application implemented using the invention needs to activate an application or service based on the legacy technologies, the address must be translated back into the name space of the appropriate legacy system. For example, an Internet (or Intranet) document stored within a Topic would also exist as a file within the Directory Structure of the underlying Operating System.

Given the fact that a new address may be translated back into the name space of the underlying legacy systems, it may seem that the invention is not adding any value. In this respect, it is important to remember that a Topic or a Library may be moved to a different physical location from time to time, and that Topics may be distributed to other Libraries. No matter where a specific instance of a Topic is stored, the addresses built using the Addressing Standard for its encapsulated resources are constant. This makes it very easy to build applications that use the resources encapsulated within distributed Topics. While the new addresses can be translated back into the name space of the underlying legacy systems, the results of these translations will vary from time to time.

Local/Remote Transparency

As previously described, Microsoft's Component Object Model (COM) which is used as the basis for the OLE and ActiveX protocols is based on the notion of Local/Remote Transparency. The idea is that an application should use a consistent method for integrating software components, and

should not have to make any special provisions for software components running within other processes or on remote network servers. Microsoft implements the notion of Local/Remote Transparency by associating remote software components with "Handlers" or "Proxies". While this approach works well from the perspective of an application, it makes the development and installation of a software component quite complex. The costs associated with the additional complexity that must be built into software components can be justified because software components are developed once and then used many times.

The invention uses a different method to realise the goal of Local/Remote Transparency. Each application is encapsulated within a Topic. To use a common software component, an application uses a set of Library services to reference the appropriate resources within the Topic containing the software component. If the Library determines that the referenced Topic is located within another Library, then a similar set of Network services are used to forward the request to the remote Library. Using this process, neither the application nor the software component needs to make any special provisions to support remote Topic references.

It is still possible to implement proxies and handlers within the embodiment by designing an instance of a software component that will act as a proxy for another remote instance of the software component.

In fact, the notion of Local/Remote Transparency can be taken a step further. Since the Topic ID is a globally unique identifier for a distributed application, an application can let the Library (or Network) search for an instance of a specific Topic ID. For example, an application could reference a database containing customer information by using only the related Topic ID. If an instance of the database were available within the local Library, it would be used to satisfy the reference. If the local instance of the database could not satisfy the request directly, then the local instance could act as a proxy for a remote instance of the database. If a local instance of the Topic could not be found, then the Library could search through an ordered list of corporate Libraries to find an appropriate remote instance of the database.

Platform Independence

While the Addressing Standard allows any Topic to reference the resources contained within other Topics across a global network, such references are useful only if the referenced resources belong to compatible

resource classes. A "resource class" is defined as a coherent and consistent set of system resources such as a specific set of Operating System services, a specific LAN interface, or the set of Internet (or Intranet) documents. The invention allows a single Topic to encapsulate various resources belonging to an arbitrary set of resource classes.

One of the best examples of a platform independent resource class is the set of Internet (or Intranet) documents. Anyone with an appropriate WWW browser can view any Internet (or Intranet) document regardless of the platform on which it was developed. The popularity of the WWW has clearly demonstrated the value of platform independent resource classes.

The embodiment also envisages provisions for the distribution of Topics to Libraries that are implemented on incompatible platforms (in this regard a "platform" may be described as an integrated set of resource classes). One solution is to define a platform independent standard for Topics. Such a standard is defined by limiting the resources encapsulated within a Topic to a set of platform independent resource classes. For example, a platform independent standard might require that all of the active components of a Topic be developed using Java (Sun Microsystem's platform independent language).

Another way to support the distribution of Topics to incompatible platforms is to build Topics that reconfigure themselves as they are published from one platform to another. From the perspective of the end user, such Topics would appear to be platform independent. However, by actively reconfiguring themselves for various platforms, these Topics could optimise their performance for the unique resource classes associated with each platform. For example, a Windows 95 gateway could be created by publishing an instance of a database on a mainframe Library to a Windows 95 Library.

The invention would also allow for individual Topics to be configured as virtual machines. Each library could then support several different types of virtual machines; some that were platform independent and some that were platform specific. Topics defined to run within platform independent virtual machines could be published or routed across heterogeneous platforms. Resources within the Topic would operate within a consistent virtual environment on all physical platforms. The boundary around the Topic (used to encapsulate the associated resources) would map

the internal (virtual) resources to external (real) resources. The boundary could map the Topic's internal file structure into the platform's file structure, or map the Topic's internal methods ino a set of distributed methods. For example, the OpenTopic method is designed to provide a consistent external entry point to a selective set of a Topic's internanal methods.

Library Organisation

As discussed previously, the Directory Structures which underlie most modern Operating Systems are used both to locate information and to organise information. While the Addressing Standard is an effective tool for locating and distributing information across a global network, it does not provide any mechanism for organising Topics within a Library.

It is envisaged that Libraries will be built to serve some purpose within some corporate context. The purpose and the corporate context associated with a Library define a natural organisational structure for the Topics contained within the Library at any specific point in time, but both the purpose and the corporate context for the Library may change over time. This means that there should be no restriction on how Topics are organised from Library to Library, or on how Topics are organised within a Library over time.

To this end, the invention allows Libraries to organise Topics into hierarchical structures as shown in Figure 4. The root 7 of a hierarchical structure is a Topic containing the core components and the configuration files for the Library itself (in this respect a Library is just another common software component). The remaining Topics 8, 9, 10 and 11 are organised as descendants of the root Topic (the Library) according to the purpose and the corporate context of the Library, as shown in Figure 6. The term "Ancestor" is used to identify the parent of a Topic, and the term "Direct Descendent" is used to describe a child of a Topic.

In principle, the invention allows Libraries to organise Topics into any structure (hierarchical, matrix, and so on) since the Addressing Standard does not rely on the Library structure to locate Topics. A Library could even support multiple organisational structures simultaneously in order to offer various end users a customised "view" of the Library. For example, an end user's view of a Library could exclude any Topics for which the end user did not have a sufficient level of access authority. Since the Addressing Standard does not rely on the organisational structure defined for a Library,

separate access levels could be defined for addressing Topics and for viewing organisational structures. While a Topic containing a software component might not appear in an end user's view of a Library, the end user might still be allowed to use an application that referenced the software component.

Managing Versions

There are two basic methods for managing versions. Versions may be created by copying Topics, or version control may be implemented within Topics. The idea of copying a Topic to create a new version is the rough equivalent of copying a file to create a new version. As with files, when a new version is created by copying a Topic, a considerable effort must be devoted to the coordination of the various versions of the Topic (either manually or through the implementation of a specialised application). If this approach is used, dependent relationships may be defined between the various versions of a Topic to simplify the associated coordination effort.

In most cases, a simpler alternative is to implement version control within a Topic. For example, a Topic may contain a document with version control. A specific version of the document is contained within a single compound file, or a single sub-directory. As new versions of the document are created they are stored in additional compound files or sub-directories within the same Topic. The Topic is also used to consolidate and store any comments submitted with respect to the various versions of the document. The distribution of such a Topic creates a new instance (or update an existing instance) containing the latest version of the document, the older versions of the document, and any associated comments.

Version control for software components is handled differently. A software development environment is set up as a Library that generates new Topics to contain new software components and applications. Multiple versions of each software component (or application) are then supported within each individual Topic. The distribution of such a Topic creates a new instance (or upgrade an existing instance) containing the latest version of the software component (or application). Using this approach, all of the source code, executable modules, documentation, and installation procedures for a specific software component are encapsulated within a single Topic. This approach simplifies the development of test environments and improves component reliability since the Topic ID and

names associated with each software component is constant from its first implementation through to its distribution.

Network Security

Since most modern Operating Systems define a single name space for applications, it is effectively impossible for an Operating System to tell the difference between a legitimate application and a virus (or other malicious program). The construction of LAN's and other networks that extend the name space of an Operating System to include various network resources, offers additional opportunities for the propagation of malicious programs. The traditional solution to this problem is to build rigid fire-walls that carefully monitor and control the flow of information between the different sections of a network. The problem with fire-walls is that they are only effective to the extent that they restrict the flow of information across a network boundary. It is inevitable that end users, frustrated by the restrictions imposed by a fire-wall, will from time-to-time circumvent the fire-wall and introduce a malicious program into the protected sections of any network.

Java (Sun Microsystems platform independent language) is an example of an Internet (or Intranet) technology that builds a fire-wall within an Operating System. Java is an interpreted language that requires the interpreter installed on each platform to include a security fire-wall. The expanding popularity of Java shows how the implementation of security barriers within an Operating System can facilitate information exchange. The problem with Java is that the fire-wall implemented within the Java interpreter is only effective for Java applications. The Java fire-wall cannot be used to isolate malicious applications written in other languages.

The invention supports an alternative method for the implementation of security barriers within Operating Systems that works equally well for all sorts of applications. Libraries could identify Topics either as "Trusted Topics" or "Suspect Topics". A Topic will be identified as a Trusted Topic if it was distributed from a Trusted Topic on a Trusted Library (or otherwise identified as a Trusted Topic by an authorised end user). Each Library maintains a short list of Trusted Libraries from which it accepts the distribution of trusted applications and software components. The set of Trusted Libraries defined by a Library is the Trusted Domain for the Library. Any Topic that was not distributed from a Trusted Topic on a

Trusted Library is identified as a Suspect Topic. A Suspect Topic is allowed to modify the resources contained within its own name space and within the name spaces of any Topics that lists the Suspect Topic as a Named Dependent. However, the Suspect Topic is not allowed to update the resources contained in any other Topic. These restrictions effectively minimise the damage done to a Library, or to a Network by a malicious Topic.

It is also possible to identify some trusted applications as Suspect Topics to contain known security exposures within an application. For example, a WWW browser implemented as a Suspect Topic could enable support for active components (Java, ActiveX, and so forth) that pose a known security risk. By designating the WWW browser as a Suspect Topic, the security risks would be limited to the name space associated with the WWW browser (which could easily be deleted and re-constructed if it was maliciously corrupted).

The approach to network security outlined above largely eliminates the need for fire-walls. Because applications and software components are isolated within individual Topics, there is less need to restrict the flow of information across intra-corporate and inter-corporate network boundaries. Even if a malicious program were to find its way into a Trusted Topic on a workstation, its malicious actions would be limited to that specific workstation since a workstation would not ordinarily be part of the Trusted Domain for any other workstation.

Basic Topic Methods (Functions)

Various instances of a Topic can be integrated to form distributed applications. One approach to installing a distributed application is to install each instance of the Topic independently, and then to configure the various instances to form a distributed application. However, such a manual installation process involves considerable time, expense, and technical expertise.

The ability to automate the construction of distributed applications is achieved by allowing each Topic to manage its own distribution. To this end, a traditional object oriented approach is used to manage the distribution of Topics. In addition to the resources encapsulated within Topics, each Topic implement a set of functions (called "Methods") to control its own distribution. These functions manage the replication of the Topic within a

Library, and the distribution of Topics to other Libraries. Methods are implemented for Topics to automate the following basic activities:

- Copy Topic
- Move Topic
- Delete Topic
- Open Topic
- Publish Topic
- Restore Topic
- Update Topic
- Route Topic

The specific methods required to implement these basic activities are defined in the following sections. The term QNID is used to describe a Compound Address.

It is not necessary for all of the Topic Methods to be defined for every Topic. For example, the instances of a specific Topic may define a distributed application designed to manage a workgroup forum. While it should be possible to copy the instance of the Topic that contains the forum, there is no reason to copy an instance that merely provides a point of access for the forum since it would not make sense to have two points of access to the same forum within a single Library.

It is also important to remember that all of the methods and Library services used by a Topic are executed within the context of that specific Topic. It may initially seem odd to execute common Library functions (or Operating System functions) within the context of an individual Topic.

Common Library functions could be implemented separately as Library methods. The problem with this approach is that the various instances of a Topic are contained in different Libraries. If the common Library functions were implemented separately, each instance of the Topic would have to identify the appropriate Library before executing a Library function. This would add an unnecessary level of complexity to every Topic. An even greater level of complexity would be required if an application wanted to execute a common Library function associated with another Topic such as a software component. Most of this complexity can be avoided by allowing Topics to inherit Library functions (and Operating System functions) from their associated Libraries.

Copy Topic

Each Topic in a network may act as a component of a distributed application, and manage its own distribution. Because of the unique characteristics of a Topic, new Topics cannot be created by a Library without some sort of template. The "Copy Topic" method allows any existing Topic to be used as a template for the creation of a new Topic.

For example, consider a Topic that manages a sales forum for a specific project. The Topic is dependent on a common software component that supports many different types of forums, and contains an appropriate set of configuration parameters for a sales forum. The Copy Topic method implemented for the sales forum produces a sales forum for another sales effort. The new sales forum is also dependent on the common software component, and inherits the appropriate configuration parameters from the existing sales forum.

Common software components could also be used as a template for one or more classes of Topic. Since there is no reason to keep two copies of a software component within a single Library, the Copy Topic method implemented for a software component could be used to produce dependent Topics. Continuing the example listed above, the Copy Topic method for a common software component could create several different classes of forums. During the execution of the Copy Topic method the end user is asked to select a specific class of forum. If the end user selects a sales forum, then the new Topic is initialised with the default configuration parameters for a sales forum.

The Copy Topic method is implemented in two parts:

```
BOOL CopyTopic(                 const char* cQNID,
                                char* cNewQNID,
                                DWORD
iSizeofNewQNID );
BOOL InitAfterCopyTopic(        const char* cOldQNID );
```

Where a cQNID is the QNID of the Ancestor for the new Topic, cNewQNID is a buffer to hold the QNID generated by the Copy Topic method for the new Topic, iSizeofNewQNID is the length of the cNewQNID buffer, and cOldQNID is the QNID of the original Topic.

To copy a Topic, the Copy Topic method defined for the Topic is called. The Copy Topic method creates a new Topic and initialises the

contents of the new Topic. Finally, the InitAfterCopyTopic method defined for the new Topic is called to complete the initialisation process.

Move Topic

The Move Topic method is used to change the hierarchical structure associated with a Library. The Move Topic method is implemented in two parts:

```
BOOL MoveTopic( const char* cQNID);
BOOL UpdateAfterMoveTopic( const char* cOldQNID);
```

Where a cQNID is the QNID of the new Ancestor for the Topic, and cOldQNID is the QNID of the Topic that was moved.

To move a Topic, the Move Topic method implemented for the Topic is called. The Move Topic method removes the Topic as a direct descendent of its current ancestor, and installs the Topic as a direct descendent of the Topic identified by cQNID. The UpdateAfterMoveTopic method is then called for all of the Topics that were moved. Note that when a Topic is moved, all of its descendants are also moved.

Delete Topic

The Delete Topic method is used to delete a Topic and all of its descendants. The Delete Topic method is implemented in two parts:

```
BOOL BeforeDeleteTopic(          void );
BOOL DeleteTopic(                const char* cQNID);
```

Where cQNID is the QNID of the Topic to be deleted.

To delete a Topic, the DeleteTopic method implemented for the Topic's ancestor is called. The BeforeDeleteTopic function is then called for all of the Topics to be deleted. If no errors are found, the Topic and all of its descendants are then deleted. Topics do not delete themselves because many Operating Systems do not allow a function to delete the file from which the function was loaded. Topics can still use the BeforeDeleteTopic method to perform any required processing before they are deleted. By returning an error code, the BeforeDeleteTopic method can also be used by a Topic to block its own deletion.

Open Topic

The Open Topic method is used to activate the application that is associated with a Topic. For example, the Open Topic method for a document with version control could activate a application designed to manage the various versions of the document.


```
BOOL OpenTopic(                  DWORD dwType,
                                 QN_LPEOLE_REQ eole_req,
                                 QN_LPEOLE_INT eole_int );
```

Where dwType is the method to be used to open the Topic, eole_req is a structure which describes how the Topic should be opened (eole_req will vary based on the value of iMethod), and eole_int is a structure in which the pointer(s) to the opened Topic are returned (eole_int will vary based on the value of iMethod).

Publish Topic

The Publish Topic method is used to distribute a Topic to another Library. The Publish Topic method assumes that the original instance of the Topic should remain on the source Library and that a copy should be distributed to the target Library. The Publish Topic method is actually implemented in four parts:

```
BOOL BeforePublishTopic(         QN_NEW_TOPIC_DATA* );
BOOL PublishTopic(               const DWORD iMethod,
                                 const char* cLocation,
                                 const char* cQNID,
                                 TCHAR* cNewLocation,
                                 DWORD iSizeofNew
                                   Location);
BOOL InitAfterPublishTopic(      QN_NEW_TOPIC_DATA* );
BOOL UpdateAfterPublishTopic(    QN_NEW_TOPIC_DATA* );
```

Where iMethod is the method to be used to publish the Topic, cLocation is the location of the target Library (cLocation will vary based on the value of iMethod), cQNID is the QNID of the default ancestor for the published Topic, cNewLocation is the location of the Topic that was created or updated by the PublishTopic method, iSizeofNewLocation is the size of the buffer allocated for cNewLocation, and QN_NEW_TOPIC_DATA is a structure defined as follows:

```
typedef struct {
    DWORD       iMethod;
    char        cQNID[QN_QNID_SIZE];
    BOOL        bExistingTopic;
    char        cName[QN_QNID_SIZE];
    char        cTargetLocation[QN_DIR_SIZE];
```

```
char        cSourceLocation[QN_DIR_SIZE];
char        cQNIDAncestor[QN_QNID_SIZE];

} QN_NEW_TOPIC_DATA;
```

Where bExistingTopic is TRUE if an instance of the Topic exists on the target Library and FALSE is an instance does not exist on the target Library, cName is the name of Topic on the target Library (this name may be different from the name of the original Topic), cTargetLocation is the location associated with the instance of the Topic in the target Library, cSourceLocation is the location associated with the instance of the Topic in the source Library, and cQNIDAncestor is the QNID of the ancestor to the Topic on the target Library (this would be the default ancestor if iExistingTopic is FALSE or the ancestor of the existing instance if iExistingTopic is TRUE).

To publish a Topic, the PublishTopic method implemented for the Topic is called. The PublishTopic method assumes that the PublishTopic methods for each Named Dependent have already been executed. The PublishTopic method will then check the target Library to see if an instance of the Topic already exists. If an instance of the Topic already exists on the target Library then the BeforePublishTopic method implemented for the existing instance is called; the contents of the existing Topic are updated; and UpdateAfterPublishTopic method for the updated Topic is called.

If an instance of the Topic cannot be found on the target Library, then an instance of the Topic is added to the target Library; the InitAfterPublishTopic method is called for the new Topic; and then the UpdateAfterPublishTopic method is called. The InitAfterPublishTopic function is used to add the new Topic to the target Library. A Library cannot update the configuration files of another Library directly since different Libraries may be implemented on different platforms and may use different formats for their configuration files.

Restore Topic

The Restore Topic method is identical to the Publish Topic method except that it restores a Topic to a previous version instead of updating the Topic to a more recent version. The Restore Topic method is designed to work with the Publish Topic method so that Topics may be backed-up and

recovered by "publishing" the Topics to another library and then "restoring" them as necessary. For example, if the original instance of a Topic were corrupted or deleted, the Restore Topic method could be used to regenerate the original instance from a published instance. The Restore Topic method will not be supported when a published instance of a Topic contains insufficient information to regenerate the original instance. The Restore Topic method is actually implemented in four parts:

```
BOOL BeforeRestoreTopic(        QN_NEW_TOPIC_DATA* );
BOOL RestoreTopic(              const DWORD iMethod,
                                const char* cLocation,
                                const char* cQNID,
                                TCHAR* cNewLocation,
                                DWORD
iSizeofNewLocation);
BOOL InitAfterRestoreTopic(     QN_NEW_TOPIC_DATA* );
BOOL UpdateAfterRestoreTopic(   QN_NEW_TOPIC_DATA* );
```

Where iMethod is the method to be used to restore the Topic, cLocation is the location of the target Library (cLocation will vary based on the value of iMethod), cQNID is the QNID of the default ancestor for the restored Topic, cNewLocation is the location of the Topic that was created or updated by the RestoreTopic method, iSizeofNewLocation is the size of the buffer allocated for cNewLocation, and QN_NEW_TOPIC_DATA is a structure in the format specified for the Publish Topic method.

Update Topic

The Update Topic method is used to control the automatic distribution of updates for Topics. When the Update Topic method is executed, the Topic checks to see if an update is available for distribution. If UpdateTopic returns TRUE then iMethod, cLocation and cQNID identify the remote instance of the Topic containing the update so that the Publish Topic method may be executed to complete the update process. The Update Topic method is implemented as follows:

```
BOOL UpdateTopic(       DWORD* iMethod,
                        char* cLocation,
                        DWORD iSizeofLocation,
                        char* cQNID,
                        DWORD iSizeofQNID);
```

Where iMethod is the method to be used to restore the Topic, cLocation is the location of the source Library (cLocation will vary based on the value of iMethod), iSizeofLocation is the size of the buffer allocated for cLocation, cQNID is the QNID of the source Topic, and iSizeofQNID is the size of the buffer allocated for cQNID.

Route Topic

The Route Topic method is used to distribute a Topic to another Library. The Route Topic method assumes that the original instance of the Topic should be moved to the target Library, and a copy of the Topic should be left on the source Library.

```
BOOL BeforeRouteTopic(      QN_NEW_TOPIC_DATA* );
BOOL RouteTopic(                const DWORD iMethod,
                                const char* cLocation,
                                const char* cQNID,
                                TCHAR* cNewLocation,
                                DWORD iSizeofNewLocation);
BOOL InitAfterRouteTopic(       QN_NEW_TOPIC_DATA* );
BOOL UpdateAfterRouteTopic(     QN_NEW_TOPIC_DATA* );
```

Where iMethod is the method to be used to route the Topic, cLocation is the location of the target library (cLocation will vary based on the value of iMethod), cQNID is the QNID of the default ancestor for the routed Topic, cNewLocation is the location of the Topic that was created or updated by the RouteTopic method, iSizeofNewLocation is the size of the buffer allocated for cNewLocation, and QN_NEW_TOPIC_DATA is a structure in the format specified for the Publish Topic method.

To route a Topic, the RouteTopic method implemented for the Topic is called. The RouteTopic method assumes that the PublishTopic methods for each Named Dependent have already been executed. The RouteTopic method will check the target Library to see if an instance of the Topic already exists. If an instance of the Topic already exists on the target Library then the BeforeRouteTopic method implemented for the existing instance is called; the contents of the existing Topic are updated; and UpdateAfterRouteTopic method for the updated Topic is called.

If an instance of the Topic cannot be found on the target Library, then an instance of the Topic is added to the target Library; the InitAfterRouteTopic method is called for the new copy; and then the

UpdateAfterRouteTopic method is called. The InitAfterRouteTopic function is used to add the new Topic to the target Library. A Library cannot update the configuration files of another Library directly since different Libraries may be implemented on different platforms and may use different formats for their configuration files.

Compound Addresses

A global address is defined for each instance of a Topic as the combination of a Library ID, and a Topic ID. Two forms of global addresses are defined: a Relative QNID in which the Library ID defaults to the local Library, and an Absolute QNID in which the Library ID is stated explicitly. In addition to these global addresses, addresses are defined in two other basic formats: Relative Addresses and Named Dependents. Compound Addresses are formed by chaining together global addresses, Relative Addresses and Named Dependents.

While the organisational structure defined for a Library should not be used to locate specific resources, there are situations in which a Topic must be able to identify other Topics in the active view. A Relative Address may be used for this purpose

Perhaps the most common use of Relative Addresses will be for the generation of Internet (or Intranet) documents. Relative Addresses can be used to generate Internet documents that support end user navigation through the active views defined for a Library. The following Relative Addresses will be defined for hierarchical views:

<-Topic> - The active Topic.

<-Ancestor> - The ancestor for the Topic in the active view.

<-Library> - The root Topic of the active view.

Note that there is no way to define a Relative Address for a direct descendent since there is generally more then one direct descendent defined for each Topic.

A Topic is said to be dependent on a second Topic if the second Topic is defined by the first Topic as a Named Dependent, where a Named Dependent is a relationship between a name and a compound address. The name associated with a Named Dependent may be used as an address. Given the above definitions for a global address, a Relative Address, and a Named Dependent, Compound Addresses may be built as follows:

- <Compound Address> = <global address>:
- <Compound Address> = <Relative Address>:
- <Compound Address> = <Named Dependent>:
- <Compound Address> = <Compound Address><Compound Address>

When two Compound Addresses are chained together, the second Compound Address is evaluated within the context of the first Compound Address. The following examples will help to illustrate how Compound Addresses are evaluated:

- **<Topic>:<Ancestor>:** - references the ancestor of the active Topic.
- **<Named Dependent>:<Ancestor>:** - references the ancestor of a Named Dependent.
- **<Ancestor>:<Named Dependent>:** - references a Named Dependent defined by the active Topic's ancestor.

For the methods presented above, any valid compound address can be used as a QNID.

A Topic has been defined as an encapsulated set of system resources. Each resource within a Topic has a name that is only valid within the context (or name space) defined by the Topic. For example, each Topic could contain a file named "index.htm" that served as an entry point for its associated Internet (or Intranet) documents. In order to link one Topic to another, the Topic references the "index.htm" file contained in the other Topic. Resources in other Topics are referenced by using the compound address for the Topic as a prefix. For example, a link to the Internet (or Intranet) documents associated with a Topic's ancestor is defined as follows: **<ancestor>:index.htm.**

Special Purpose Resource Classes

Three special purpose resource classes will be defined for the initial implementations: Named Dependents, Tags, and Named Files.

As previously discussed, a Named Dependent is a relationship between a name and a Compound Address. The Compound Addresses identified as Named Dependents describe a set of Topics on which the active Topic is dependent. For example, a Topic containing a document ordinarily includes a word processing system as a Named Dependent.

A Tag is a relationship between a name and a short text phrase. Tags are used primarily by applications to define short text phrases that are to be

inserted into the Internet (or Intranet) documents generated for a Topic. The following standard Tags will be defined for all Topics:

- **Name** - the name of the Topic.
- **<Compound Address>** - the name of the specified Topic.
- **Author** - the name of the individual that produced the Topic.
- **Address** - the mailing address for the Author.
- **E-mail Address** - the e-mail address for the Author.
- **Company** - the Company that developed the Topic.

A Named File is a relationship between a name and a path. In a full implementation, all of the files associated with a Topic will be encapsulated within the Topic and there would be no need for Named Files. However, when the invention is implemented on top of a legacy system, there may be a need to include one or more files within a Topic without actually moving the files. In this regard, a Named File identifies a file that is encapsulated within a Topic but has not yet been embedded within the Topic's File System. Copies of any Named Files are embedded within the remote instances of the Topic, when the Topic is Published or Routed. The following method will be used to embed the Named Files defined for a Topic:

```
BOOL EmbedNamedFiles(          void          );
```

Name Space Methods

In a full implementation, name space management is a fairly simple matter since there is no need to translate names. However, implementations that are built on top of legacy systems will need to continually convert names from the Topic's name space to one or more legacy name spaces. Name translations are made even more complex since legacy names (such as LAN based Directory Structures) often vary according to a specific end user's workstation configuration.

Four methods will be implemented for Topics to manage names:

```
BOOL GetNamedItem(      const char* cRClass,
                        const char* cTopicName,
                        const char* cReferenceName,
                        char* cValue,
                        DWORD iSizeofValue);
BOOL GetFirstNamedItem( const char* cRClass,
                        const char* cReferenceName,
```

```
                                    char* cTopicName,
                                    DWORD iSizeofName,
                                    char* cValue,
                                    DWORD iSizeofValue);
    BOOL GetNextNamedItem( const char* cRClass,
                                    const char* cReferenceName,
                                    char* cTopicName,
                                    DWORD iSizeofName,
                                    char* cValue,
                                    DWORD iSizeofValue);
    BOOL SetNamedItem(      const char* cRClass,
                                    const char* cTopicName,
                                    const char* cValue);
```

Where cRClass is the name of a specific Resource Class, cTopicName is the name of a resource defined in the Topic's name space, iSizeofName is the size of the buffer defined for cTopicName, cReferenceName is required as a reference point for some name space translations, cValue is the value associated with cTopicName, iSizeofValue is the size of the buffer defined for cValue.

For the initial implementations, the following resource classes will be supported:

Relative QNID - the short form QNID used within Libraries.

Absolute QNID - the long form QNID used within Networks.

Tag - relationships between names and short text phrases.

Named Dependent - relationships between names and dependent Topics.

Direct Descendent - a direct descendent associated with a Topic.

Named File - relationships between names and files.

Relative Path - a path for a file that is relative to a reference path.

Absolute Path - a fully qualified path for a file.

The GetNamedItem and SetNamedItem methods will not be defined for a Topic's Direct Descendants. Also, the SetNamedItem method will not be defined for Relative Paths, Absolute Paths, Relative QNID's, or Absolute QNID's. At least for the initial implementations, the items returned by GetFirstNamedItem and GetNextNamedItem will not include items with standard names. For example, the items returned by GetFirstNamedItem

and GetNextNamedItem for the "Tag" resource class will not include the Name, Address, E-mail Address or any other standard Tag defined for a Topic.

Internet (or Intranet) Documents

As previously discussed, the initial implementations will include the set of Internet (or Intranet) documents as a platform independent resource class. This will allow Libraries to be used as platform dependent Intranet publishing tools. The idea is that users with no special expertise in Intranet technologies could produce Intranet documents as a natural by-product of their day-to-day activities. Also, since Topics can contain applications of arbitrary complexity, the implementation should not reduce the productivity of these users.

One approach is to require each Topic to produce its own Internet (or Intranet) documents. While such an approach would always be an option for a Topic, it would add an unnecessary level of complexity to each Topic. Further, the use of different methods to produce Internet (or Intranet) documents in each Topic would make it difficult for authorised end users to change the look and feel of the Internet (or Intranet) documents produced by each Topic.

To solve this problem, there is a default method for the generation of Internet (or Intranet) documents. This method requires each Topic to include a set of Styles. Styles are sets of Internet (or Intranet) documents that include Pre-processor Commands. Each Library includes a pre-processor that integrates Library organisation, Topic Styles, and content to produce a set of Internet (or Intranet) documents for a Topic.

The Pre-processor Commands are defined as follows:

- **<XO Tag="Name of a Tag">** - this command is replaced by the value of the Tag by the pre-processor. If the "Name of a Tag" is a Compound Address, the command is replaced by the Name of the referenced Topic.
- **<XO NamedFile="Name of a Named File">** - this command is replaced with the relative path to the Named File by the pre-processor.
- **<XO AbsolutePath="Path to a File">** - this command is replaced with the absolute path to the specified file. The "Path to a File" may

be a reference to a file in another Topic (by using a compound address as a prefix).

- **<XO RelativePath="Path to a File">** - this command is replaced with the relative path to the specified file. The "Path to a File" may be a reference to a file in another Topic (by using a compound address as a prefix).
- **<XO Text="Path to a File">** - this command indicates that another text file should be inserted within the Styles. This command allows applications to generate large sections of text (that may contain additional Pre-processor Commands) for insertion within the Topic Styles.
- **<XO AllDirectDescendents>** - this command is replaced with a series of list items. Each list item includes a reference to one of the direct descendants defined for the Topic.
- **<XO AllNamedDependents>** - this command is replaced with a series of list items. Each list item includes a reference to one of the Named Dependents defined for the Topic.
- **<XO AllNamedFiles>** - this command is replaced with a series of list items. Each list item includes a reference to one of the Named Files defined for the Topic.

The initial implementations will automatically run the pre-processor for each Topic when the contents of the Topic are updated, or the organisation of the Library is changed. This approach will produce a set of static Internet (or Intranet) documents that may be viewed at any time by existing Internet (or Intranet) browsers. This is a satisfactory approach given that the initial implementations will only support a single view for each Library. It would be impossible to coordinate all the different versions of the static Internet (Intranet) documents that would be required to support multiple views.

An alternative approach supports multiple views. One such alternative approach, called Dynamic Pre-processing, requires that the pre-processor be run as individual documents are accessed across the network. Since the pre-processor could then be run within the context of a specific view, the Internet (or Intranet) documents could easily be customised to that specific view.

To support the generation of static Internet (or Intranet) documents, the following methods will be implemented for each Topic:

```
BOOL ProcessStyles( void );
BOOL UpdateStyles( void );
BOOL UpdateTemplate( void );
```

The ProcessStyles method takes the styles that have been defined for a Topic and uses them to generate a set of Internet (or Intranet) documents for the Topic. The UpdateStyles method replaces the styles that have been defined for a Topic with the default styles for that class of Topic. The default styles are stored with the Template that was used to create the Topic. The UpdateTemplate method replaces the Templates associated with a Topic with the default Templates for that class of Topic. These Templates are used to create and update Topics through the CopyTopic and PublishTopic methods.

The invention allows the Styles and Templates for a Topic to be customised to meet the specific requirements of an individual author. The UpdateStyles and UpdateTemplate methods provide a means to undo these custom changes. The UpdateStyles and UpdateTemplate methods also provide a means to distribute enhanced Styles and Templates that would not otherwise be distributed through the PublishTopic method.

Although the invention has been described with reference to a preferred embodiment it should be appreciated that it could be embodied in other forms. For instance, the invention could be embodied as a general purpose corporate network, a public network (similar in scope to the World Wide Web (WWW)), or a network of low cost dynamically reconfigured workstations (commonly called "Network Computers").

It will be appreciated by persons skilled in the art that numerous variations and/or modifications may be made to the invention as shown in the specific embodiments without departing from the spirit or scope of the invention as broadly described. The present embodiments are, therefore, to be considered in all respects as illustrative and not restrictive.

THE CLAIMS DEFINING THE INVENTION ARE AS FOLLOWS:-

1. A distributed operating system for a network, comprising:

a collection of libraries each of which exists at some physical location and is associated with a collection of instances of different topics;

each Topic is an encapsulated set of system resources and has a file system containing applications, documents, databases, software components, or any other collection of logically related files, only one instance of a Topic may exist in any given Library at any given time, but instances of the same Topic may exist in different Libraries; wherein

an Addressing Standard which requires a first and second globally unique identifier is applied to the Libraries and Topics, the first globally unique identifier is used to identify each Library, and the second globally unique identifier is used to identify each Topic.

2. A distributed operating system according to claim 1, wherein every instance of a Topic will have the same Topic identifier no matter which Library it is in.

3. A distributed operating system according to claims 1 or 2 wherein the Topics are organised into hierarchical structures with a root Topic and descendant Topics, the hierarchical structures are used only to locate Topics, and the globally unique identifier is used exclusively to identify Topics.

4. A distributed operating system according to claim 1, 2 or 3 wherein each Topic implements a set of object oriented methods to copy a Topic within a Library, to open a Topic, and to publish (or route) a Topic to another Library.

5. A distributed operating system according to claim 4 wherein each Topic implements additional object oriented methods to restore a Topic from a published instance of the Topic, and to control the automatic distribution of updates for Topics.

6. A distributed operating system according to claim 1, 2, 3, 4, or 5 wherein a consistent communication protocol exists to enable interaction between the applications, software components, and documents stored as different Topics within a single Library or within different Libraries on a Network.

7. A distributed operating system according to claim 1, 2, 3, 4 or 5 wherein applications and compound documents are built by combining sets of standard and custom components that are distributed as separate Topics.

8. A distributed operating system according to claim 4 or 5 wherein distributed applications are constructed by using the object oriented methods implemented by individual Topics to distribute data and software components across a network.

9. A distributed operating system according to claim 1, 2, 3, 4 or 5 wherein the Topics within a Library are organised into multiple views which are used only to locate Topics and not to identify Topics.

10. A distributed operating system according to claim 1, 2, 3, 4 or 5 wherein a global address is defined for each instance of a Topic as the combination of a Library ID, and a Topic ID.

11. A distributed operating system according to claim 1, 2, 3, 4, or 5 wherein a Relative Address is used by a Topic to identify other Topics in the active view.

12. A distributed operating system according to claim 1, 2, 3, 4, or 5 wherein a Topic is said to be dependent on a second Topic if the second Topic is defined by the first Topic as a Named Dependent.

13. A distributed operating system according to claim 10, 11 or 12, wherein Compound Addresses are formed by chaining together global addresses, Relative Addresses and Named Dependents.

14. A distributed operating system according to claim 1, 2, 3 4,or 5 wherein styles are defined for each Topic so that an appropriate set of Internet (or Intranet) documents may be generated to share the contents of the Topic with an extended workgroup.

15. A distributed operating system according to claim 1, 2, 3, 4 or 5 wherein access to individual Topics is restricted to authorised users.

16. A distributed operating system according to claim 1, 2, 3, 4 or 5 wherein a Topic is identified as a Trusted Topic if it is published from a Trusted Topic contained in a Trusted Library and each Library maintains a list of Trusted Libraries from which it accepts the distribution of trusted applications and software components; any Topic that is not specifically identified as a Trusted Topic is considered to be a Suspect Topic.

17. A distributed operating system according to claim 16, wherein Suspect Topics are only allowed to modify the resources contained within

their own name space and within the name spaces of any Topics that list the Suspect Topic as a Named Dependent; Suspect Topics are not allowed to update the resources contained in any other Topic.

18. A distributed operating system according to claim 1, 2, 3, 4 or 5 wherein a template is defined for each Topic that controls how the content and behaviour of the Topic can mutate in useful ways as the Topic is copied within a Library, or published to another Library.

19. A distributed operating system according to claim 1, 2, 3, 4 or 5 wherein each Topic runs in a virtual machine, and the boundary around each Topic maps internal, or virtual, resources to external, or real, resources.

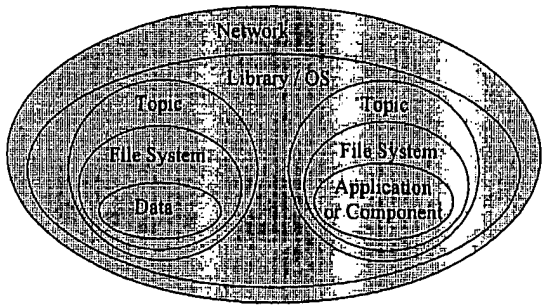


Figure 1

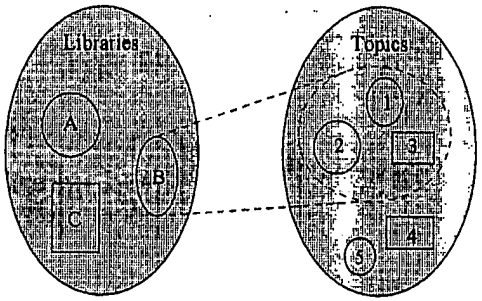


Figure 2

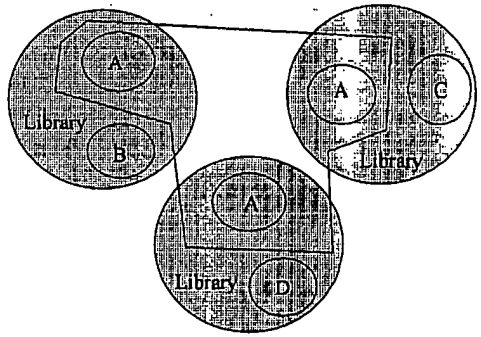
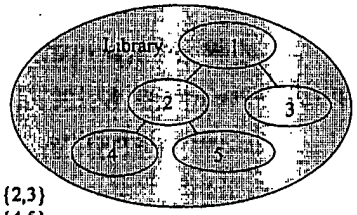


Figure 3



1 is the Ancestor of {2,3}
2 is the Ancestor of {4,5}
{2,3} are Direct Descendents of 1
{4,5} are Direct Descendents of 2

Figure 4

INTERNATIONAL SEARCH REPORT

International Application No.
PCT/AU 97/00840

A. CLASSIFICATION OF SUBJECT MATTER

Int Cl[6]: G06F 15/16

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC G06F 9/45, 15/16, 17/30

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
AU: IPC as above

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
WPAT: distrib: and operat: syst: or o/s
INSPEC: distrib: and operat: syst: or o/s and address: and global:

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	Proceedings of the Second International Workshop on object Orientation in Operating Systems, PP 96-99, Published: Los Alamitos, CA, USA, 1992 IEEE Computing Society Press "Distribution + Persistence = Gbbal Virtual Memory, A Position Paper" whole document	
A	Proceeding of the 12th International Conference on distributed Computing Systems PP 581-588, published: Los Alamitos, CA, USA, 1992, IEEE Computer Society Press "Naming & Addressing of Objects without Unique Identifiers" whole document	

[X] Further documents are listed in the continuation of Box C

[X] See patent family annex

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier document but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
16 January 1998	02 FEB 1998
Name and mailing address of the ISA/AU AUSTRALIAN INDUSTRIAL PROPERTY ORGANISATION PO BOX 200 WODEN ACT 2606 AUSTRALIA Facsimile No.: (02) 6285 3929	Authorized officer S. KAUL Telephone No.: (02) 6283 2182

Form PCT/ISA/210 (second sheet) (July 1992) coplma

INTERNATIONAL SEARCH REPORT

International Application No.
PCT/AU 97/00840

C (Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 5471615 A (AMATSU et al) 28 November 1995 whole document	
A	WO 92/21186 A (BELL ATLANTIC NETWORK SERVICES INC), 26 November 1992 whole document	

INTERNATIONAL SEARCH REPORT
Information on patent family members

International Application No.
PCT/AU 97/00840

This Annex lists the known "A" publication level patent family members relating to the patent documents cited in the above-mentioned international search report. The Australian Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

Patent Document Cited in Search Report		Patent Family Member			
US	5471615	JP	5233570		
WO	92/21186	AU	79545/91	NZ	242527

END OF ANNEX